Replanning for Situated Robots

Michael Cashmore and **Andrew Coles**
King's College London

Bence Cserna
University of New Hampshire

Erez Karpas
Technion — Israel Institute of Technology

Daniele Magazzeni
King's College London

Wheeler Ruml
University of New Hampshire

Abstract

Planning enables intelligent agents, such as robots, to act so as to achieve their long term goals. To make the planning process tractable, a relatively low fidelity model of the world is often used, which sometimes leads to the need to replan. The typical view of replanning is that the robot is given the current state, the goal, and possibly some data from the previous planning process. However, for robots (or teams of robots) that exist in continuous physical space, act concurrently, have deadlines, or must otherwise consider durative actions, things are not so simple. In this paper, we address the problem of replanning for situated robots. Relying on previous work on situated temporal planning, we frame the replanning problem as a situated temporal planning problem, where currently executing actions are handled via Timed Initial Literals (TILs), under the assumption that actions cannot be interrupted. We then relax this assumption, and address situated replanning with interruptible actions. We bridge the gap between the low-level model of the robot and the high-level model used for planning by the novel notion of a *bail out action generator*, which relies on the low-level model to generate high-level actions that describe possible ways to interrupt currently executing actions. Because actions can be interrupted at different times during their execution, we also propose a novel algorithm to handle temporal planning with time-dependent durations.

Introduction

The connection between planning and execution is a seemingly simple one: a robot will typically first plan, that is, come up with a course of action that achieves its goal, and then execute that plan. However, planning is usually done using a relatively low fidelity model, in order to make the planning process tractable. This can lead to the not-infrequent need to replan. Other possible causes for replanning include newly sensed information, failure to execute some action (while other actions might still continue executing), or the goal changing.

The typical view of replanning is that the robot is given its current state, its goal, and possibly some data from the previous planning process (e.g., the previous plan or the generated search nodes). If the robot actually existed in a world that obeyed the assumptions of classical planning, then this would indeed be correct. However, for robots (or teams of robots) that exist in continuous physical space, act concurrently, have deadlines, or must otherwise consider durative actions, things become more complex. Consider, for example, a durative encoding of a MOVE(A,B) action, which appears in many planning domains. This action is meant to model a robot moving from waypoint A to waypoint B. It is usually modeled as deleting at(A) in the beginning, and adding at(B) at the end. Now consider the state of the world, and specifically the location of the robot, if replanning is triggered while the MOVE action is executing. Since at(A) was already deleted, but at(B) was not yet added, the robot is currently nowhere. This discrepancy is caused by the fact that the robot is actually located at some real set of coordinates, but planning uses an abstraction of the real world consisting of waypoints. This mismatch will cause a naive replanning process to fail.

In this paper, we propose a general solution to this problem. Our overall approach is to frame the replanning problem as planning with a *dynamic* initial state. This dynamic initial state is modeled by a temporal planning problem, which captures the effects and conditions of currently executing actions using timed initial literals (TILs) (Cresswell and Coddington 2003; Edelkamp and Hoffmann 2004).

Our first treatment of this problem assumes that actions are non-interruptible and that, once an action has started, it will either continue executing until it succeeds or it will fail and trigger replanning. We show how previous work on situated temporal planning (Cashmore et al. 2018) can be used to handle the dynamic initial state described above, and thus is also useful for replanning while actions are currently executing.

However, forcing an agent to complete an action it has started can be highly suboptimal. For example, consider a rover driving to waypoint A to sample a rock there. Suppose halfway through the drive, the agent realizes that the drill is stuck, making drilling impossible, and thus invalidating the plan and triggering replanning. Forcing the rover to finish the drive to A is pointless, as it no longer has any actions to perform at A. Even worse, this wastes time and consumes battery energy, which could have been used to achieve other goals. Thus, we will also address the case where actions can be interrupted.

As we previously implied, interrupting an action requires

a higher fidelity model of the world, which makes modeling difficult and planning intractable. Thus, we assume only carefully limited use of such a model, in the form of a black-box 'bail out action' generator that, based on a more detailed model of the agent's state, can propose various ways to bail out of a currently executing action. For example, a bail out action generator for move actions might generate paths from the real position of the robot to several nearby waypoints. Importantly, bailout actions are *local*, in that they only describe ways to bail out of a currently executing action. Thus, instead of planning with one high-fidelity model of the world, which could be difficult to come up with, we can plan with one low-fidelity model combined with several models for bail out actions, which do not necessarily have to be consistent with each other or account for *all* aspects of the problem.

Recall that time still passes while we are replanning, and the robot's real position when replanning is triggered may be different from its real position when the new plan will start executing, and thus the duration of the bail out action will be different depending on when it begins execution. However, as we explain later, these *time-dependent durations* obey a certain monotonicity property, namely that the ending time of the bailout action is monotonic in its start time — that is, the earlier the bailout action is triggered, the earlier it completes. We present a novel algorithm for checking the consistency of a Simple Temporal Network (STN) (Dechter, Meiri, and Pearl 1991) with such monotonic time-dependent durations, which we have integrated with the OPTIC planner (Benton, Coles, and Coles 2012).

We evaluate our approach on a real robotic testbed — a Turtlebot robot in an office delivery task, as well as in the Robocup Logistics League (RCLL) simulation (Niemueller, Lakemeyer, and Ferrein 2015). Our results show that our replanning approach outperforms a baseline which can only plan from a static state, either when there are deadlines in the problem, or when there are multiple agents which our approach can better exploit.

Related Work

Action failures, goal changes, and other online execution issues have long been facts of life in the planning community. One way to cope is to produce plans in a representation with inherent flexibility, such as representing exact execution times as a temporal constraint network (Ghallab and Laruelle 1994), or to return partial policies, such as contingent plans (Pell et al. 1997; Myers 1999). When the only uncertainty is about action durations, planning can be combined with scheduling under uncertainty to come up with robust plans (Cimatti et al. 2018). However, trying to account for uncertain outcomes often impedes scalability.

Another way to handle change is to replan. For example, when plans for their printing system are invalidated by broken machine modules, Ruml et al. (2011) attempt to rescue in-flight sheets by replanning the remainder of their trajectories. This requires estimating the time required by replanning, so that an appropriate state can be chosen at which the new plan will replace a suffix of the original plan.

Planning online is also explored in Domain Predictive Control (Löhr et al. 2013; 2014), which combines planning with Model Predictive Control to generate control input sequences for hybrid dynamical systems. Perhaps the most closely related work to ours is the IxTeTeXEC executive (Lemai and Ingrand 2004), which interleaves planning and execution, and takes into consideration the fact that execution will start only after planning finishes.

Fox et al. (2006) contrast replanning with plan repair, and argue that the latter can promote plan stability, the similarity of a new plan to the original. In applications involving multi-agent coordination or multi-level planning, stability can provide benefits. Cushing and Kambhampati (2005) provide a counter-argument, arguing that plan repair can be subsumed by replanning where, among other changes, the objective of the new planning problem might penalize deviating from the commitments of the previous plan. This allows an agent to properly balance the costs and benefits of using actions similar to those in the original plan. All of this previous work assumes that failures conveniently happen at action boundaries and that the world stops while the agent replans.

Cashmore et al. (2018) consider planning problems in which actions depend on externally timed events, such as taking a bus, and the time taken during planning can be long enough to affect the choices of the planner. For example, if planning is anticipated to take a long time, plans involving taking the next bus may not be found soon enough to be executable. They develop a strategy in which externally timed events are modeling using TILs and estimates of remaining planning time are used to prune infeasible heuristic search nodes in the planner.

Preliminaries

We consider propositional temporal planning problems with Timed Initial Literals (TIL) (Cresswell and Coddington 2003; Edelkamp and Hoffmann 2004). Such a planning problem Π is specified by a tuple $\Pi = \langle F, A, I, T, G \rangle$, where:

- F is a set of Boolean propositions that describe the state of the world.
- A is a set of durative actions. Each action $a \in A$ is described by:
  - Minimum duration $dur_{\min}(a)$ and maximum duration $dur_{\max}(a)$, both in $\mathbb{R}^{0+}$ with $dur_{\min}(a) \leq dur_{\max}(a)$,
  - Start condition $cond_{\vdash}(a)$, invariant condition $cond_{\leftrightarrow}(a)$, and end condition $cond_{\dashv}(a)$, all of which are subsets of F, and
  - Start effect $eff_{\vdash}(a)$ and end effect $eff_{\dashv}(a)$, both of which specify which propositions in F become true (add effects), and which become false (delete effects).
- $I \subseteq F$ is the initial state, specifying exactly which propositions are true at time 0.
- T is a set of timed initial literals (TIL). Each TIL $l = \langle time(l), lit(l) \rangle \in T$ consists of a time $time(l)$ and a literal $lit(l)$, which specifies which proposition in F becomes true (or false) at time $time(l)$.

- $G \subseteq F$ specifies the goal, that is, which propositions we want to be true at the end of plan execution.

A solution to a temporal planning problem is a schedule σ, which is a sequence of triples $\langle a, t_a, d_a \rangle$, where $a \in A$ is an action, $t_a \in \mathbb{R}^{0+}$ is the time when action a is started, and $d_a \in [dur_{\min}(a), dur_{\max}(a)]$ is the duration chosen for a. A schedule can be seen as a set of instantaneous *happenings* (Fox and Long 2003) that occur when an action starts, when an action ends, and when a timed initial literal is triggered. Specifically, for each triple $\langle a, t, d \rangle$ in the schedule, we have action a starting at time t (requiring $cond_{\vdash}(a)$ to hold a small amount of time ϵ before time t, and applying the effects $eff_{\vdash}(a)$ right at t), and ending at time $t + d$ (requiring $cond_{\dashv}(a)$ to hold ϵ before $t + d$, and applying the effects $eff_{\dashv}(a)$ at time $t + d$). For a TIL l we have the effect specified by $lit(l)$ triggered at time $time(l)$. Finally, in order for a schedule to be valid, we also require the invariant condition $cond_{\leftrightarrow}(a)$ to hold over the open interval between t and $t+d$, and that the goal G holds at the state which holds after all happenings have occurred.

Interruptible and Non-Interruptible Actions

As previously mentioned, actions might be interruptible or non-interruptible. A high-level propositional model does not necessarily capture this difference. Consider two actions, both of which move an object from waypoint A to waypoint B:

Ballistic Launch: The action is implemented by launching an object using a catapult.

Driving: The action is implemented by driving.

Both of these actions can be modeled with $pre_{\vdash} = at(A)$, $eff_{\vdash} = \neg at(A)$, $eff_{\dashv} = at(B)$, with a controllable duration in the range $[d_{min}, d_{max}]$. However, for the ballistic launch action, the duration is chosen when the action is executed (by setting the appropriate trajectory), and can not be changed during execution — that is, the action is non-interruptible.

On the other hand, for the driving action, the duration is, in fact, a different way of representing the average velocity. If this action is interrupted, the duration can be adjusted, or the agent can even choose to drive to any other location. However, all of these options are not modeled in the standard propositional representation of this action.

In the next section, we explain how we address situated replanning with only non-interruptible actions, by relying on previous work on situated temporal planning (Cashmore et al. 2018). Then, we describe one way of modeling interruptible actions, and how we can replan with interruptible actions for situated agents, that is, online.

Replanning with Non-interruptible Actions

We now describe our approach for replanning during execution with non-interruptible actions. For example, consider the well known match cellar domain, in which a fuse must be fixed while a match is burning. The LIGHT-MATCH action serves as an envelope for the FIX-FUSE action; that is, FIX-FUSE must be executed within the time interval when

$$
\begin{aligned}
F' =& F \cup \{e_a \mid \langle a, t_a, d_a \rangle \in CA\} \\
A' =& \{a' \mid a \in A\}, \text{where :} \\
& dur_{\min}(a') = dur_{\min}(a) \\
& dur_{\max}(a') = dur_{\max}(a) \\
& cond_{\vdash}(a') = cond_{\vdash}(a) \cup \\
& \quad \{e_b \mid \langle b, t_b, d_b \rangle \in CA, del_{\vdash}(a) \cap cond_{\leftrightarrow}(b) \neq \emptyset\} \\
& cond_{\leftrightarrow}(a') = cond_{\leftrightarrow}(a) \\
& cond_{\dashv}(a') = cond_{\dashv}(a) \cup \\
& \quad \{e_b \mid \langle b, t_b, d_b \rangle \in CA, del_{\dashv}(a) \cap cond_{\leftrightarrow}(b) \neq \emptyset\} \\
& eff_{\vdash}(a') = eff_{\vdash}(a) \\
& eff_{\dashv}(a') = eff_{\dashv}(a) \\
I' =& I \\
T' =& \{\langle time(l) - t, lit(l) \rangle \mid l \in T, time(l) \geq t\} \cup \\
& \{\langle t_a + d_a - t, f \rangle \mid \langle a, t_a, d_a \rangle \in CA, f \in eff_{\vdash}(a)\} \cup \\
& \{\langle t_a + d_a - t, e_a \rangle \mid \langle a, t_a, d_a \rangle \in CA \\
G' =& G
\end{aligned}
$$

Figure 1: Replanning Task Definition

LIGHT-MATCH is executing. Now assume that FIX-FUSE failed for some reason, but we have already lit our only remaining match. Thus, we must come up with a plan and be able to execute it within the remaining time until LIGHT-MATCH has finished.

One decision we make here is that, when replanning has been triggered, we never start executing a new action from the original schedule π — we only finish executing the currently executing actions. This is because we do not yet know whether future actions in the (stale) plan might lead to a dead end, in which case we will never be able to solve the problem. Thus, we choose to possibly err on the side of inaction, and deal only with the currently executing actions. We are also assuming here that, if action failure triggered replanning, the failed actions had either no effect on the state or that the effects of the failure are fully observed and modeled at the start of replanning.

In general, let $\Pi = \langle F, A, I, T, G \rangle$ be the planning problem, and assume we are in the process of executing a schedule π when replanning is triggered at time t, when the current state of the system is s. As previously mentioned, it would be incorrect to simply replan from state s, because (a) there might be some actions which have already started but have not yet ended, and their invariant conditions and end effects must be taken into consideration, and (b) the TILs in Π need to be updated to reflect that the current time is t.

We will denote the set of currently executing actions occurrences by $CA = \{\langle a, t_a, d_a \rangle \in \pi \mid t_a \leq t < t_a + d_a\}$, that is, the action occurrences in the schedule which have started but not yet ended. We now describe a temporal planning problem $\Pi' = \langle F', A', I', T', G' \rangle$ which captures the *dynamic* state we are planning from — namely, that the currently executing actions will finish.

Π', described fully in Figure 1, is very similar to Π, except that it contains a fact e_a for each currently executing action, which indicates that the action has finished. The e_a facts work together with TILs (as explained below) and slightly modified action definitions to enforce the invariant conditions of currently executing actions, as we now explain.

The TILs in Π' capture three different things. First, we adjust the timing of the TILs from the original planning task, to reflect that the current time is t, so past TILs are removed, and the time for future TILs is adjusted by decreasing it by t. Second, we use TILs to encode the end effects of currently executing actions, so that if action a will finish in d time units, its effects are encoded as TILs which will occur at time d. Finally, we use TILs to signal that an action has ended (by setting e_a to true), and therefore its invariant condition no longer has to hold. This, in turn, enables all snap actions (start or end of a durative action) which delete the invariant condition of a — which is implemented by adding e_a as a condition for all snap actions whose effects delete some fact in $cond_{\leftrightarrow}(a)$.

We can now use previous work on situated temporal planning (Cashmore et al. 2018) to solve the above planning task. Since this technique accounts for time passing while planning is going on, the solution returned by the planner will be executable when the planner terminates — even if the currently executing actions have not finished yet.

To continue the above example, suppose the duration of LIGHT-MATCH is 10 seconds, and FIX-FUSE failed 2 seconds after LIGHT-MATCH started. We encode this using a TIL, stating that the end effect of LIGHT-MATCH (i.e., the light going out) will occur 8 seconds from when replanning started. Essentially, this imposes a deadline of 8 seconds on goal achievement time (GAT) — that is, planning time + plan makespan, which the situated temporal planner can account for.

Replanning with Interruptible Actions

We now turn our attention to the problem of replanning with interruptible actions. As previously mentioned, we must first overcome the issue of modeling what are the possible ways to bail out of a currently executing action. We assume we have access to a bail out action generator, which takes as input an action (that is assumed to be currently executing — we still do not *start* executing actions from the stale plan before a new plan is found), and returns as output a list of *possible* bail out actions. The bail out action generator can be thought of as extending the successor generator with alternative endings of the currently executing action. This bail out generator can be implemented as a black box, and will run every time replanning is triggered. Thus, the bail out action generator can have access to lower level state variables than the high-level propositional model used for planning does.

Although the details of bail out actions are domain-specific, there are common properties that we discuss. Essentially, the most important property of a bail out action is that, after the bail out action has ended, the high-level propositional planning model is *consistent*. In other words, the bail out action needs to correctly specify the high-level propositional state of the world *after* the bail out action has finished, which ensures that after applying a bail out action, we can keep planning using the more abstract high-level propositional model.

For example, consider the previous example of a rover driving from waypoint A to waypoint B to sample a rock, when the drill breaks at some point during the drive. The bail out action generator can utilize a path planner, which has access to the rover's real position (and possibly even to the underlying dynamical model which accounts for mass, acceleration, obstacles, turning radius, ...). These bail out actions restore the state of the world to a consistent one, where after the bail out action finishes the rover has a location, instead of not being located anywhere *during* the drive.

When modeling bail out actions for high-level propositional planning, a good rule of thumb is that a bailout action for a should have something to do with the state variables a affects. In light of the abovementioned discussion on restoring consistency, this means that when we model a bail out action for a, we should start by looking at the effects of a, and identify facts which are made inconsistent by the start effects of a, whose consistency is restored by the end effects of a. Our bail out action should fix those, but of course, the low-level details of the domain could dictate other preconditions and effects for the bail out actions.

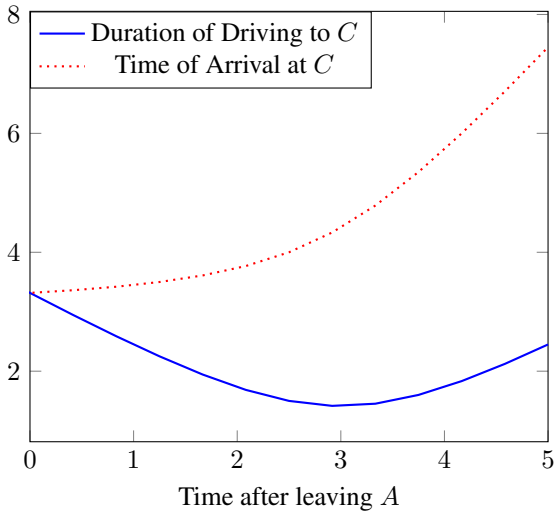
An additional complication occurs when considering the fact that multiple actions could be executing currently, and bailing out of them might not be independent. For example, consider a rover that is driving from A to B, and taking pictures of the route from A to B — which can be modeled using two concurrently executing actions: drive(A, B) and photo(A, B). In this case, bailing out of drive(A, B) also requires bailing out of photo(A, B). If we have such co-dependent actions, our bail out generator could take as input sets of currently executing actions, and generate options to bail out of all of the input actions. As implied by the above discussion, such co-dependency will typically occur only for actions which have some shared fact in their preconditions or effects.

Durations of Bail Out Actions

We now turn our attention to the durations of the bail out actions. Continuing with our rover example, consider the map illustrated in Figure 2a, where the rover is driving from A to B, and replanning is triggered when it is in the position drawn in the figure, about one third of the way from A to B. One possible bail out action is to drive to C. The duration of this bail out action depends on the position of the rover, which in turn depends on the time the bail out action starts. Thus, in order to plan in such a scenario, we must extend our planner to handle time-dependent durations. Furthermore, this time-dependent duration could be *non-monotonic*. In our example, the duration decreases until the rover reaches the point on the line from A to B where it is closest to C, and starts increasing after passing that point, as illustrated in Figure 2b (the solid blue line).

Previous work on situated temporal planning (Cashmore et al. 2018) relied on estimating the latest time a plan π

(a) Map of Waypoints



(b) Time-dependent Duration

Figure 2: Illustration of Rover Example

can start, denoted $estimated\_latest\_start(\pi)$. This estimate relied on both the current plan π as well as the Temporal Relaxed Planning Graph (TRPG) (Coles et al. 2010) from the state reached by π. Unfortunately, with non-monotonic time-dependent durations, it is not enough to look at a single number to describe the latest time a plan can start.

Note, however, that the *ending time* of the bailout action does increase monotonically with the time the bail out action started, as shown in the dotted red line in Figure 2b. We claim this is always the case, under some reasonable assumptions, which we discuss next.

Optimal Bailout Actions and Monotonicity

Recall that our bailout action generator has access to a higher fidelity model of the world, although it plans for a shorter horizon. If this planner returns optimal plans, then the ending time of the bailout action should always increase monotonically with when the bail out action started.

In our rover example, where we ignore obstacles, the shortest path to C is a straight line. The rover travels on the straight line from A to B for some distance d, and then bails out and drives in a straight line towards C. Clearly, the *total distance* the rover travels on the way to C, and thus the duration, increases with d, although not necessarily linearly.

To formalize this intuition, assume that each high-level action a corresponds to a plan $\pi_a = \langle l_1, l_2, \ldots, l_m \rangle$ in some higher-fidelity (low-level) model of the world, where l_i are low-level actions. We also assume the bailout action generator uses an optimal planner for this low-level model, minimizing the total duration of the plan. Finally, assume the agent bails out of a after executing $\langle l_1, l_2, \ldots, l_n \rangle$, and denote the low-level state of the world at this time by x_n.

Now consider what happens if the agent bails out earlier, after $n' < n$ low-level actions. Note that we are assuming the same bailout target, that is, the same goal in the low-level model in both cases. We will denote this bailout goal by b_g. It is easy to see that the optimal plan from $x_{n'}$ to b_g can not be more expensive than any plan which continues the execution of a until l_n and then bails out to b_g, that is, a plan from $x_{n'}$ to b_g which is constrained to start with $\langle l_{n'+1}, \ldots, l_n \rangle$. Thus, if our bailout action generator relies on an optimal planner, we automatically get monotonicity in the ending time of the bailout action.

However, it is also easy to show examples where this property does not hold. If our bailout action generator calls a sampling-based motion planner, e.g., RRT (Lavalle, Kuffner, and Jr. 2000), then we can not guarantee that it returns an optimal path. Nevertheless, if we ever get such low level plans that bailing out at time t_1 finishes later than bailing out at time t_2, but $t_1 < t_2$, we can always improve upon the bail out action at time t_1, by continuing execution of the original action until time t_2, and bailing out then. In other words, because it is always possible to bail out later, we can obtain monotonicity in the ending time of bail out actions via simple post processing of the durations of the bail out actions, which chooses the best time to bail out *after* some time point, instead of exactly at that time point.

We have seen the monotonic nature of the ending time of bail out actions. We now explain how we exploit this property in checking the temporal consistency of partial plans with time-dependent durations, which is integrated into our temporal planner.

Temporal Planning with Time-Dependent Durations

In prior temporal planners (Coles et al. 2010; 2009), the temporal constraints on a plan under construction have been represented as a Simple Temporal Problem (Dechter, Meiri, and Pearl 1991) – a collection of constraints, each $lb \leq t_j - t_i \leq ub$, with $ub, lb \in \Re_0^+$, recording the constraints on the temporal separation of the plan steps t_i and t_j. State progression updates an STP stored in each state, according to the actions applied. Then, by solving the STP, one can determine whether the temporal constraints are consistent; and if so, obtain the minimum and maximum time at which each step can be scheduled. In the nominal case, in temporal planning, only the minimum times are used when reporting the time-stamps of a solution plan.

In the case where actions' durations are known at the point the action is applied, during search, an STP is sufficient: the duration of an action is encoded as a constraint between its start and end step indexes, with a known lower- and upper-bound. In the case where actions' durations are time-dependent, though, this is no longer the case: the duration of the action is not known until the time at which it is to

Algorithm 1: STP with Time-Dependent Durations

Data: STP constraints T and the corresponding plan of snap-actions P
Result: ts, the timestamps of the plan steps in P; or $\bot$ if the STP is inconsistent

1 $T_{td} \leftarrow \emptyset$;
2 **do**
3 $\quad ts \leftarrow$ solve to find minimum timestamps of the STP $(T \cup T_{td})$;
4 $\quad$ **if** $ts = \bot$ **then return** $\bot$;
5 $\quad converged \leftarrow \top$;
6 $\quad T_{td} \leftarrow \emptyset$;
7 $\quad$ **foreach** *start–end snap action pair* $\langle a_\vdash, a_\dashv \rangle \in P$, *with step indexes* i *and* j *respectively* **do**
8 $\quad\quad dur \leftarrow$ duration of a if started at time $ts[i]$;
9 $\quad\quad min\_dur \leftarrow$ minimum time-dependent duration of a at any time at-or-after $ts[i]$;
10 $\quad\quad max\_dur \leftarrow$ maximum time-dependent duration of a at any time at-or-after $ts[i]$;
11 $\quad\quad$ add $ts[i] + dur \leq t_j$ to T_{td};
12 $\quad\quad$ add $min\_dur \leq t_j - t_i \leq max\_dur$ to T_{td};
13 $\quad\quad$ **if** $\neg(dur \leq ts[j] - ts[i] \leq dur)$ **then**
14 $\quad\quad\quad converged \leftarrow \bot$;
15 $\quad$ **end**
16 **while** $\neg converged$;
17 **return** ts

be applied is known.

To address this issue, we take a two-fronted approach. First, during search, when applying an action a with a time-dependent duration that starts at step i and finishes at step j, the STP is constrained so a can have any duration between its global minimum and global maximum – across any of the times for which its duration is defined. This admissibly relaxes the time-dependency into a duration interval.

Second, when checking that the temporal constraints in a given state are consistent – formerly a case of simply checking the consistency of the STP – an iterative refinement process is used, presented in Algorithm 1, which exploits the monotonicity requirement on the durations of bailout actions.

The algorithm takes as input the STP T from the state, as well as the plan P that reached it, and builds a collection of additional STP constraints T_{td} to capture the time-dependent durations. On the first iteration, these are empty, hence at line 3 the STP solved is that from the state – if this is inconsistent, the algorithm terminates immediately.

Otherwise, the algorithm then inspects the times given to actions, to see what time-dependent durations are relevant – assuming they started at-or-after the times in ts, with $ts[i]$ denoting the time of step i. For each start–end action pair in the plan, with step indexes i and j, we look up three values:

- What the duration of the action would be if it started *exactly at* time $ts[i]$ – the duration dur
- What the minimum possible duration of the action would be, assuming it has to come *at or after* time $ts[i]$ – the duration min_dur
- What the maximum possible duration of the action would be, assuming it has to come *at or after* time $ts[i]$ – the duration min_dur

These values are used in two ways. First, 'feasibility cuts' are added to T_{td}, as follows:

- if dur is indeed the duration the action should have, the earliest time step j could occur is $ts[i] + dur$ (line 11).
- in any case, starting the action at or after time $ts[i]$ bounds its duration between min_dur and max_dur (line 12).

Second, the value of dur is used to determine whether the current solution to the STP is consistent with respect to the time-dependent durations. If this is not the case (i.e. $ts[j] - ts[i] \neq dur$ – line 13) the $converged$ flag is set to false – indicating that the algorithm must continue to iterate.

The correctness of this algorithm exploits the monotonicity requirement of bailout actions; i.e. that delaying the start of a bailout action *may* reduce its duration, but *may not* lead to a net reduction in the time at which the action can end. If this were not the case, then search would be needed at line 11: we would lose the ability to determine an admissible lower-bound on the end time of an action based on its start time and the duration it would have at that time, as a better end-time may be attainable by delaying the start.

The algorithm is guaranteed to converge due to the behavior of the feasibility cuts. For the cuts delaying the ends of actions, the cuts on one iteration dominate those in the previous iteration: the ends of actions are iteratively delayed, but never made earlier. Delaying the ends of actions may in turn delay the starts of other actions; and increasing the start time of an action only ever reduces the range $[min\_dur, max\_dur]$ until there is only one option left – i.e. the duration is no longer time-dependent. Then, as $dur = min\_dur = max\_dur$, the cut added to T_{td} on line 12 guarantees that on subsequent iterations, the duration constraint will always be satisfied. At the limit, all actions are assigned their latest possible time-dependent duration, so none of the duration constraint checks at line 13 will lead to $converged$ being set to $\top$; so the loop terminates.

Empirical Evaluation

In order to empirically evaluate our approach to replanning, we evaluated it on problems from the Robocup Logistics League (RCLL) Simulation (Niemueller, Lakemeyer, and Ferrein 2015; Niemueller et al. 2016), and on a real robotic system. We compare our approach to a baseline which can only replan from a static state. Thus, we formulate a planning problem whose initial state is the state that will hold when all currently executing actions finish. However, we use a situated temporal planner (Cashmore et al. 2018) which can plan while the currently executing actions finish. We force this planner to only start executing after all currently executing actions finish by adding a new fact e (which stands for execute), which is added by a TIL when all currently executing actions finish, and is a start condition of all actions.

We remark that an even more naive baseline, which does not require situated temporal planning, would be to only

#	1 Robot		2 Robots		3 Robots	
	R	B	R	B	R	B
5	9.635	9.635	0.05	0.35		
15	45.158	45.654		100.961	54.258	77.46
18	15.55	15.551	34.131	38.1	1.041	1.041
28	63.998	60.527		144.902	49.325	49.325
32	62.267	56.488	60.331		16.79	18.102
34	129.301	83.464	57.541	73.139		124.393
35	82.102	82.429	63.517	47.817	31.401	31.401
39		55.526		179.377	162.422	
52	60.78	41.672				
55	18.299	18.3	132.501	136.862		
56	52.746	42.526	51.004	57.145		
57	43.959	43.948		69.007	5.942	5.942
64	41.899	46.577	16.698	26.688	100.72	165.269
68	59.974	59.256		123.063	108.989	74.078
71	60.992	50.979			9.564	9.564
73	22.76	22.76	44.677	85.833		84.632
78		52.162	222.93			112.392
80	0.803	0.803	46.24	46.24	71.078	72.462
81	66.478	98.285	54.74		27.62	27.62
86	134.195	97.714	95.733	84.247		155.941
89	12.038	12.038	134.84	48.361		146.077
91	133.342	127.479	55.047	76.08	52.872	69.099
99	66.826	65.996				
SOLVED	21	23	15	17	13	17
Avg. GAT	56.34	51.53	60.99	60.07	44.13	50.11

Table 1: Goal Achievement Time on RCLL Instances for Baseline (B) and Our Replanning Approach (R) for Instances with no Deadline

#	1 Robot		2 Robots		3 Robots	
	R	B	R	B	R	B
5	1.791	1.791	0.05	0.28	10.456	10.456
15	8.702	8.702	57.258	80.907		149.369
18	55.39	54.412	57.189	97.039	66.225	
28	44.316	45.669		149.862	51.997	83.426
32	34.751	34.751			46.474	78.681
34	36.054	36.054	64.219	69.711	59.348	57.063
35	114.534	116.784		48.013		68.772
39	67.44	45.394		67.384	56.848	
52	43.267	47.299		108.064	56.795	59.125
55	66.745	56.732				
56	18.947	18.948	86.399	97.877	36.816	71.843
57		46.221	26.152	26.152	59.144	59.144
64	92.747	111.351	91.723	65.061		
68	49.312	51.866			32.655	32.655
71	106.72	99.832			72.467	
73		52.142	72.753	72.285		125.065
78	8.756	8.756	55.687	70.239		146.216
80	59.327	58.357	79.032	76.492		
81	132.003	61.396	51.084	58.399		70.166
86	2.707	2.707			17.911	17.912
89	42.057	46.896	4.144	4.144	33.411	33.411
91	114.602	123.41	53.322	60.394		68.341
99	50.697	50.697	84.643		86.475	106.005
SOLVED	21	23	14	17	14	17
Avg. GAT	54.8	51.51	53.77	59.92	44.68	55.42

Table 2: Goal Achievement Time on RCLL Instances for Baseline (B) and Our Replanning Approach (R) for Instances with Deadline

start planning after all currently executing actions have finished. However, this is clearly inferior to the baseline we used, and thus we omit it from the experiment.

We now describe each of these experiments in detail.

RCLL

The Robocup Logistics League Challenge presents a planning problem where 3 robots must fulfill orders that arrive dynamically by feeding workpieces to 6 different machines. Each machine can perform one type of processing step, and fulfilling an order requires performing a series of processing steps on different machines. Each order can also have a deadline — we ran one experiment on a version with deadlines and another on a version without deadlines.

In order to trigger replanning, we first generate a plan to solve the original problem. Then we randomly choose some point in the plan (using a uniform distribution), choose one of the machines at random (also uniformly), and trigger replanning by simulating a failure of the chosen machine. Unlike in the original challenge, we assume that when the machine fails, it will be fixed at some known time in the future, which is also chosen randomly between 10% to 30% of the makespan of the original plan. As we do not modify robot behaviors here, we did not implement a bail out action generator, and we assume actions are non-interruptible.

For this experiment, we started with 100 random RCLL instances generated for a previous paper (Schaepers et al. 2018). For each of these problems we have 3 versions: with 1, 2, or 3 robots. We only used the 23 instances which were solved within our time limit of 200 seconds for all 3 versions of the problem. We triggered replanning for all 3 versions of each of these 23 instances, and compare our proposed approach here with the baseline.

The experiments were run on a virtual machine running on powerful laptop (Intel Core-i7 8750H CPU). 6 planners were run in parallel using the parallel tool (Tange 2011), each with a time limit of 200 seconds and no memory limit (the VM had 16 GB of virtual memory).

The results for problems without a deadline are described in Table 1 and those for the version with a deadline in Table 2. Both of these tables show the goal achievement time (GAT) from when replanning was triggered, as well as the total number of problems solved and the average GAT (which is taken over the instances solved by both approaches for each number of robots).

The picture in both tables is similar and as expected: for 1 robot, there is no benefit in our replanning approach. This is because each robot can only execute a single action at any given moment, and thus there is no benefit to using our more sophisticated replanning approach. The overhead of our approach, which creates a larger planning problem, is also evident here with the smaller number of problems solved. However, as the number of robots increases, our approach benefits, as it has more opportunities to start executing actions with one or two robots while the others complete their currently executing actions.

Real Robotic System: Office Delivery

In our second experiment, we used a real robotic system, where a Turtlebot 2 robot serves as a delivery robot in an office environment. In the scenario, the robot is used to transport objects between areas. Deliveries must be completed within deadlines, and the robot must monitor its battery level and recharge when it becomes too low (this triggers replanning with the added constraint that the robot must immedi-

Problem	1	2	3	4	5	6	7	8
Bailout	777.568	824.573	841.183	620.600	339.760	514.008	795.625	901.881
Baseline	-	-	-	-	-	-	-	-

Problem	9	10	11	12	13	14
Bailout	1587.022	1177.209	1177.207	1656.577	1061.984	1098.133
Baseline	-	1169.432	-	-	1136.626	-

Table 3: GAT for problems in the Office Delivery scenario. Times in seconds. "-" denotes that the problem was made unsolvable due to deadlines passing while waiting for actions to complete.

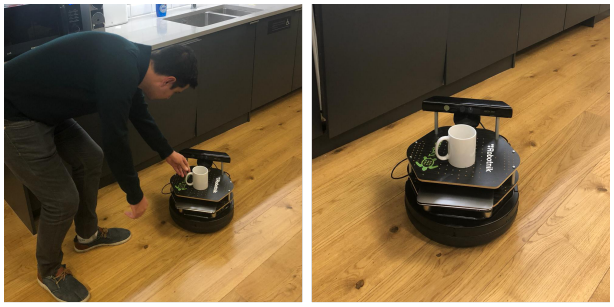


Figure 3: The Turtlebot 2 platform used in the second experiment to perform delivery tasks.

ately recharge).

The domain for the robot includes navigation actions between waypoints, actions for requesting and waiting for human assistance, and assisted pick and place actions. In addition the domain includes actions for docking to recharge, undocking, and localizing the robot, which must be performed after each recharge and before navigation can begin. Replanning typically occurs while the robot is navigating between two locations and the charge level drops below the replanning threshold.

As this is a real robotic system, we also implemented a bail out action generator for the navigation actions the robot performs. The bailout action generator for moving from A to B along some path p generates a bailout action of going to some other waypoint C. This is implemented as follows:

1. Sample points along the current path p at intervals of 4 meters.
2. From each sampled point x, a collision-free path is computed by the default global planner of the ROS navigation package. This path planner applies Dijkstra's algorithm to the global 2D costmap.
3. The duration of the bailout action is time-dependent: we compute the time of arrival t_i to each sampled point x_i along the original path p. While the time from when move(A, B) started is between t_{i-1} and t_i, the duration of bailing out to C is the sum of t_i and the duration of following the computed path from x_i to C. We remark that even though the path planner is not optimal, we did not have to perform the post processing described above to enforce monotonicity of the ending times.

Using the bail out action generator allows the robot to decide, when replanning is triggered, to cancel the currently executing action and instead execute the bailout action.

To perform this experiment, we integrated the approach we describe here, as well as the new planner which supports time-dependent durations, with ROSPlan (Cashmore et al. 2015). As ROSPlan has already been integrated with the Turtlebot 2, no extra effort was required there. The only programming required for integrating the research described here with ROSPlan was extending the system such that the bailout action generator described above is called every time replanning is triggered (by ROSPlan, this time). To do this the ROSPlan *Problem Interface* was extended to call the bailout generator. This node generates the PDDL problem instance to be solved by the planner. The planning problem that is generated is solved by calling the new planner. To integrate the new planner, no changes needed to be made.

The goal achievement time is shown in Table 3. The experiment consists of 14 problems, each containing 4-5 delivery goals. The delivery destinations and deadlines were randomized, with deadlines between 3 and 25 minutes. We used two sets of problems. In the first set (problems 1 – 8, in the upper half of Table 3), there was only one possible bail out destination: the dock. In the second set of problems (problems 9 – 14, in the lower half of Table 3), there were multiple possible bail out destinations: the dock and all other charging stations throughout the office. For each problem, replanning was triggered manually immediately after dispatching the first navigation action. As these results show, using the baseline approach of waiting for the currently executing action to complete renders the problem unsolvable due to the deadlines on the delivery tasks in almost all cases.

Summary and Future Work

In this paper, we have addressed replanning for situated agents. The contributions of this paper are threefold: First, we framed the problem of situated replanning with executing actions as a temporal planning problem with a dynamic initial state, utilizing TILs. Second, we introduced the notion of bail out action generators, which bridge the gap between the high-level model used for planning and the low-level realistic model, and allow us to interrupt currently executing actions. These bail out actions lead to actions with time-dependent durations, and thus our third contribution deals with temporal reasoning with time-dependent durations.

As expected, and confirmed by our empirical evaluation, our approach is beneficial in situations where there is enough time to come up with a plan, and start executing it, before the currently executing actions finish. Furthermore, if we can

not interrupt currently executing actions, and each agent is limited to performing only one action at a time, then our approach is only beneficial if there are multiple agents, and thus one agent can start executing actions even before all the other agents have finished their actions. As the experiment on real robots shows, our approach is especially useful with long action durations.

Finally, we remark that although this paper addresses replanning, we have not made use of the previous plan. In future work, we will explore ways of exploiting the previous plan, by integrating some plan reuse or repair techniques (Fox et al. 2006) into our framework.

References

Benton, J.; Coles, A. J.; and Coles, A. 2012. Temporal planning with preferences and time-dependent continuous costs. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS)*.

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtós, N.; and Carreras, M. 2015. Rosplan: Planning in the robot operating system. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, 333–341.

Cashmore, M.; Coles, A.; Cserna, B.; Karpas, E.; Magazzeni, D.; and Ruml, W. 2018. Temporal planning while the clock ticks. In de Weerdt, M.; Koenig, S.; Röger, G.; and Spaan, M. T. J., eds., *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018, Delft, The Netherlands, June 24-29, 2018.*, 39–46. AAAI Press.

Cimatti, A.; Do, M.; Micheli, A.; Roveri, M.; and Smith, D. E. 2018. Strong temporal planning with uncontrollable durations. *Artif. Intell.* 256:1–34.

Coles, A.; Fox, M.; Halsey, K.; Long, D.; and Smith, A. 2009. Managing concurrency in temporal planning using planner-scheduler interaction. *Artificial Intelligence* 173(1):1–44.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-chaining partial-order planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*, 42–49.

Cresswell, S., and Coddington, A. 2003. Planning with timed literals and deadlines. In *Proceedings of 22nd Workshop of the UK Planning and Scheduling Special Interest Group*, 23–35.

Cushing, W., and Kambhampati, S. 2005. Replanning: a new perspective. In *Proceedings of ICAPS-05*.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49(1-3):61–95.

Edelkamp, S., and Hoffmann, J. 2004. PDDL2.2: The language for the classical part of the 4th international planning competition. Technical Report 195, University of Freiburg.

Fox, M., and Long, D. 2003. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research (JAIR)* 20:61–124.

Fox, M.; Gerevini, A.; Long, D.; and Serina, I. 2006. Plan stability: Replanning versus plan repair. In *Proceedings of ICAPS-06*, 212–221.

Ghallab, M., and Laruelle, H. 1994. Representation and control in IxTeT, a temporal planner. In *Proceedings of AIPS-94*, 61–67.

Lavalle, S. M.; Kuffner, J. J.; and Jr. 2000. Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions*, 293–308.

Lemai, S., and Ingrand, F. 2004. Interleaving temporal planning and execution in robotics domains. In *AAAI*, 617–622.

Löhr, J.; Eyerich, P.; Winkler, S.; and Nebel, B. 2013. Domain predictive control under uncertain numerical state information. In *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling, ICAPS 2013, Rome, Italy, June 10-14, 2013*.

Löhr, J.; Wehrle, M.; Fox, M.; and Nebel, B. 2014. Symbolic domain predictive control. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, 2315–2321.

Myers, K. 1999. Cpef: A continuous planning and execution framework. *AI MAgazine*.

Niemueller, T.; Karpas, E.; Vaquero, T.; and Timmons, E. 2016. Planning Competition for Logistics Robots in Simulation. In *ICAPS Workshop on Planning and Robotics (PlanRob)*.

Niemueller, T.; Lakemeyer, G.; and Ferrein, A. 2015. The RoboCup Logistics League as a Benchmark for Planning in Robotics. In *WS on Planning and Robotics (PlanRob) at Int. Conf. on Aut. Planning and Scheduling (ICAPS)*.

Pell, B.; Gat, E.; Keesing, R.; Muscettola, N.; and Smith, B. 1997. Robust periodic planning and execution for autonomous spacecraft. In *Proceedings of IJCAI-97*.

Ruml, W.; Do, M. B.; Zhou, R.; and Fromherz, M. P. J. 2011. On-line planning and scheduling: An application to controlling modular printers. *Journal of Artificial Intelligence Research* 40:415–468.

Schaepers, B.; Niemueller, T.; Lakemeyer, G.; Gebser, M.; and Schaub, T. 2018. ASP-based time-bounded planning for logistics robots. In *Proceedings of the 28h International Conference on Automated Planning and Scheduling (ICAPS)*.

Tange, O. 2011. Gnu parallel - the command-line power tool. *;login: The USENIX Magazine* 36(1):42–47.